AI & ML Final Project Report

Onkita Gope ogope@gmu.edu
Sneha Apsangi sapsangi@gmu.edu
Yemberzal Sartaj ysartaj@gmu.edu
Deekshita Sanampudi dsanampu@gmu.edu

1 Abstract

Machine learning has become a critical component of modern network intrusion detection systems as adversaries use an increasing amount of automated and AI-assisted attack techniques. Traditional signature-based detection methods struggle to generalize against evolving threats; therefore, there is a motivation towards adopting data-driven approaches that are capable of learning complex behavioral patterns from network traffic. In this project, we compare deep learning architectures for binary classification of benign versus malicious network traffic using the CIC-IDS2017 dataset. Specifically, we implement and evaluate a transformer-based Bidirectional Encoder Representations from Transformers (BERT) model adapted to tokenized packet representations and a convolutional neural network (CNN) trained on engineered network flow features. The models are trained by performing data split strategies and learning rate configurations to evaluate the generalization performance and training stability. We assess the model's performance using accuracy, confusion matrices, and receiver operating characteristic (ROC) curves. Our results demonstrate that both architectures are capable of learning meaningful representations of network behavior, with trade-offs in training complexity and detection performance. This project outline provides a practical insight into how deep learning and large language models can be applied to network intrusion detection and focuses on the key considerations for deploying ML-based security systems.

2 Introduction

The growing complexity and scale of modern computer networks and rapid advancement of adversarial attack techniques have reduced the effectiveness of rule-based and signature-driven intrusion detection systems (IDS). These conventional approaches often fail to detect evolving threats that do not match known patterns. As a result, many commercial network security platforms have incorporated Machine Learning (ML) and Large Language Model (LLM) techniques into their detection pipelines. As attackers continue to adopt automation and AI-driven tools, defenders must also rely on ML-assisted systems to maintain effective and adaptive network security.

For the scope of this project, we found publicly accessible datasets with labeled network traffic to simulate realistic network environments and include a diverse range of attack scenarios. The dataset allows for model performance but is highly dependent on data preprocessing choices, feature representation, training and testing splits, and hyperparameters. Poor configuration of these elements can result in overfitting, unstable training behavior, or misleading performance evaluations.

In this project, we implement and compare two deep learning architectures for binary classification of malicious network traffic: a BERT-based transformer model trained on tokenization representation of packet data and a CNN-based model trained on numerical network flow features. The CNN model applies convolutional layers to learn localized feature patterns within network flows, while the BERT model leverages attention mechanisms to capture contextual relationships between tokens, allowing for a more sequence-aware understanding of network behavior. Overall, this study offers practical insight into deep learning and transformer-based approaches for network traffic classification in IDS.

3 Methodology

For the first milestone of our project, we had to select a dataset for training our models. We looked at several datasets, including NSL-KDD and UNSW-NB15, but selected the CIC-IDS2017 dataset. This

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

was because of the dataset's high-quality features and how the data is separated by days. This dataset isolates different attacks by day, which allows for a targeted analysis.

For a detailed analysis, we focused on the network traffic that was captured on Thursday, July 6, 2017. Specifically, we focused on the Thursday-WorkingHours-Morning-WebAttacks, which contains web attacks. We selected this specific dataset to create a baseline for binary classification, and we aim to distinguish between normal background traffic and the malicious traffic patterns generated by Brute Force attacks. Given that the source indicated that Brute Force attacks were conducted between 9:20AM - 10:00 AM on Thursday July 6th, the CSV file was edited in respect to the timestamps to make sure only data relating to the Brute Force attacks was used.

3.1 Data Preprocessing and Textual Features

To make network traffic data compatible with BERT, we transformed the features of the network packets into structured sequences. This is because BERT expects natural language input and is pre-trained on language and text sources. This differs from the CNN model, which performs well with purely structural data.

1. **Label Binarization:** To simplify the binary classification task, 'BENIGN' activity was mapped to 0 and 'MALICIOUS' activity was mapped to 1. This was done forr both models.
2. **BERT Protocol Resolution:** Protocol numbers were mapped to their standard names to provide semantic context. We used this mapping:
   - $1 \rightarrow$ ICMP
   - $6 \rightarrow$ TCP
   - $17 \rightarrow$ UDP
   - $0 \rightarrow$ HOPOPT
3. **BERT Sequence Construction (Flow ID Plus):** To make the network data compatible with BERT, we created a feature called `Flow ID Plus`. This converts network numbers into descriptive sentences to give them semantic meaning. By combining the IPs, ports, and protocols into a string (e.g., "Destination IP... - Source IP..."), the model can interpret traffic patterns in the scope of contextual text.

Unlike the BERT model, which requires textual input, the CNN model operates directly on the numerical network flow features. The data processing workflow was designed for saving important traffic characteristics and for preventing data leakage.

1. **CNN Changes:** Then, non-numeric and identifiers like IP addresses and timestamps, were removed to prevent data leakage and the model from memorizing specific hosts or sessions. The remaining features were converted to numerical format and any missing or infinite values were replaced by zeroes for numeraical stability during training.
2. **Z-Score Normalization:** All the features were standardized using z-score normalization because CNN is sensitive to feature scale. Each network flow was then represented as one-dimensional feature vector which was then reshaped to include a single channel. The results was an input shape of (features, 1) per sample and this led to the use of one-dimensional convolutional layers to learn localized feature patterns across the flow statistics.

3.2 Model Architecture and Tokenization

For BERT, we used the Hugging Face `transformers` library.

- **BERT Tokenization:** We used the `bert-base-uncased` AutoTokenizer. The `Flow ID Plus` strings were passed through the tokenizer to generate PyTorch tensors for input IDs and attention masks.
- **BERT Architecture:** For the classification head, we used the `BertForSequenceClassification` model (specifically `bert-base-uncased`).

For CNN, tokenization was not used.

- **CNN Architecture:** he CNN architecture was implemented using TensorFlow and Keras. The architecture is made of three one-dimensional convolutional layers with increasing filter sizes of 32, 64, and 128 filters. Each of these followed by ReLU activation and max-pooling layers. This design allows the network to progressively learn higher level representations of network traffic patterns and also reducing dimensionality.

- **Output Layers:** After convolutional layers, the feature maps were flattened and passed through a fully connected layer with 128 neurons and ReLU activation. Dropout regularization was applied to reduce overfitting of the model. The final output layer uses a softmax activation function to produce class probabilities for benign and malicious traffic.

3.3 Training and Evaluation

This dataset was divided into a 70% training split and a 30% testing split. The training loop was created as follows:

- **BERT Class Weighting:** Since our dataset contained more benign traffic than malicious traffic, the model risked becoming biased toward the majority. To prevent this, we applied inverse frequency weighting, which puts a higher penalty for misclassifying malicious attacks as benign. The weight for each class c was computed as $w_c = \frac{N}{K \times \text{count}(c)}$, where N is the total number of training samples and K is the number of classes.
- **BERT Optimizer and Epochs:** We used the **AdamW** optimizer with a learning rate of 2×10^{-5}. The model was fine-tuned for **3 epochs** with a batch size of 64. A larger batch size was used for efficiency.
- **CNN Optimizer and Epochs:** During the training, 20% of the training data was also reserved as a validation set. Early stopping was also implemented based on validation loss and a maximum of 3 epochs to prevent overfitting and save the best performing model weights. The model was compiled using categorical cross-entrpy loss and trained with **Adam** optimizer using a learning rate of 2×10^{-5}.

4 Experiments and Results

4.1 Experimental Setup

The experiments were conducted using Python in a Google Colab environment. We used a GPU in order to accelerate the training process. For the dataset, we used the CIC-IDS2017 intrusion detection dataset: `https://www.unb.ca/cic/datasets/ids-2017.html`. We utilized the Hugging Face `transformers` library for model instantiation and `PyTorch` for tensor operations for BERT. For CNN, we used REPLACE WITH WHAT WE USED FOR CNN.

The model hyperparameters that were used are displayed in Tables **??** and **??**. The dataset was split into training (70%) and testing (30%).

Table 1: Hyperparameter Configurations for BERT and CNN Fine-Tuning

BERT Fine-Tuning

Parameter	**Value**
Base Model	`bert-base-uncased`
Optimizer	AdamW
Learning Rate	2×10^{-5}
Batch Size	64
Epochs	3
Loss Function	Weighted Cross-Entropy

CNN Fine-Tuning

Parameter	**Value**
Optimizer	Adam
Learning Rate	2×10^{-5}
Batch Size	64
Epochs	15
Loss Function	Categorical Cross-Entropy

4.2 Results

The model's performance was evaluated over 3 epochs, both training and testing loss were monitored for overfitting. Table **??** shows the accuracy and loss metrics recorded at the end of each epoch for the BERT model. Table **??** shows the accuracy and loss metrics for the CNN model for the first thee epochs.

4.2.1 Classification Performance

To analyze the model's ability to distinguish between Benign (0) and Malicious (1) flows, we generated a Confusion Matrix and a Receiver Operating Characteristic (ROC) curve. The Confusion Matrices (Figure 1) demonstrate the models' precision in identifying malicious flows. Figure 2 shows the trade-off between True Positive Rate and False Positive Rate in the trained models.

Algorithm 1 BERT-Based Sequence Classification

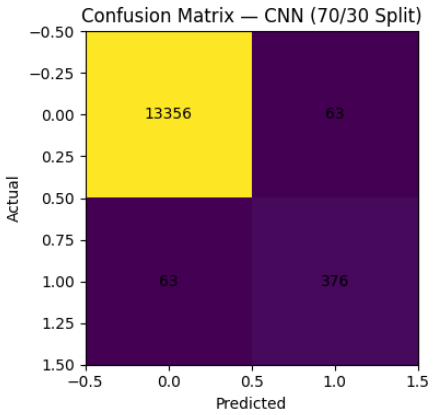
Require: Tabular Dataset $\mathcal{D}$, Pre-trained Model $\mathcal{M}_{bert}$
Ensure: Fine-tuned Classifier θ^*

Step 1: Textual Transformation
procedure FEATUREENGINEERING($\mathcal{D}$)
    $\mathcal{D}$[Label] $\leftarrow$ ($\mathcal{D}$[Label] == 'BENIGN' ? 0 : 1)
    $\mathcal{D}$[Prot] $\leftarrow$ Map({6:'TCP', 17:'UDP', 1:'ICMP', 0:'HOPOPT'})
    $\mathcal{D}$[Text] $\leftarrow$ "DstIP " $+\mathcal{D}$[DstIP] $+\cdots+$ " Label " $+\mathcal{D}$[Label]
    return $\mathcal{D}$
end procedure
Step 2: Tokenization and Weighting
procedure PREPAREDATA($\mathcal{D}_{text}, Labels$)
    $Tok \leftarrow$ AutoTokenizer('bert-base-uncased')
    $Encodings \leftarrow Tok(\mathcal{D}_{text}, \text{padding} = \text{True}, \text{truncation} = \text{True})$
    $Weights \leftarrow$ CalculateInverseFreq($Labels$)
    return $Encodings, Weights$
end procedure
Step 3: Training
procedure TRAINLOOP($Encodings, Labels, Weights$)
    Initialize $\mathcal{M}_{bert}$ ('bert-base-uncased')
    $Optim \leftarrow$ AdamW($\mathcal{M}_{bert}$.parameters, $lr = 2e-5$)
    $LossFn \leftarrow$ CrossEntropyLoss($weight = Weights$)
    for $epoch \leftarrow 1$ to 3 **do**
        $\hat{y} \leftarrow \mathcal{M}_{bert}$.forward($Encodings$)
        $Loss \leftarrow LossFn(\hat{y}, Labels)$
        $Loss$.backward()
        $Optim$.step()
    end for
    return $\mathcal{M}_{bert}$
end procedure

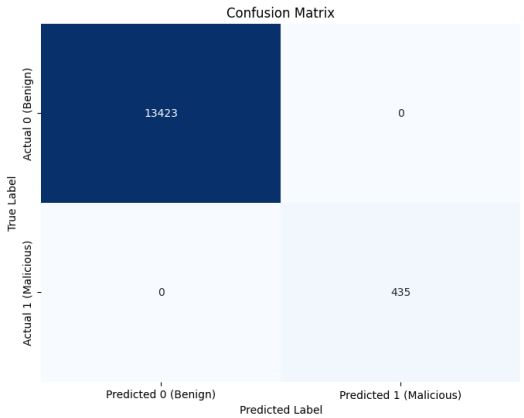
Algorithm 2 CNN-Based Network Traffic Classification

Require: Tabular Dataset D, CNN Model M_{cnn}
Ensure: Trained Classifier θ^*

Step 1: Feature Preprocessing
procedure FEATUREPREPROCESSING(D)
    D[Label] $\leftarrow$ (D[Label] == 'BENIGN' ? 0 : 1)
    Remove identifier-based features (IP addresses, timestamps, flow IDs)
    Convert remaining features to numeric values
    Normalize features using standard scaling
    Resize features into 1D CNN input format
    return Feature matrix X, Labels y
end procedure
Step 2: Dataset Preparation
procedure PREPAREDATA(X, y)
    Split X, y into training set (70%) and testing set (30%)
    Reserve a validation set from the training data
    return $X_{train}, X_{test}, y_{train}, y_{test}$
end procedure
Step 3: Training
procedure TRAINCNN(X_{train}, y_{train})
    Initialize CNN model M_{cnn}
    $Optim \leftarrow$ Adam(M_{cnn}.parameters, $lr = 2 \times 10^{-5}$)
    $LossFn \leftarrow$ CrossEntropyLoss()
    for $epoch \leftarrow 1$ **to** E **do**
        $\hat{y} \leftarrow M_{cnn}(X_{train})$
        $Loss \leftarrow LossFn(\hat{y}, y_{train})$
        $Loss$.backward()
        $Optim$.step()
        Apply early stopping based on validation loss
    end for
    return M_{cnn}
end procedure



(a) CNN Model

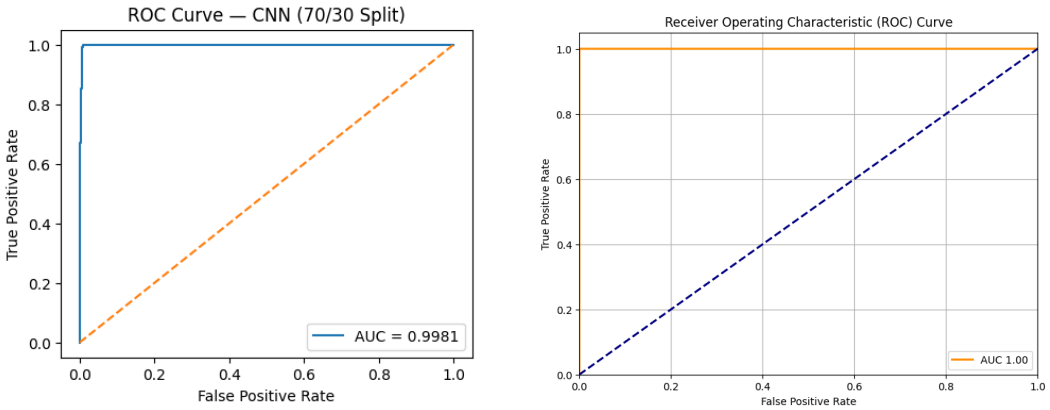


(b) BERT Model

Figure 1: Confusion Matrices showing True Positives and False Positives for the trained models.

Table 2: Training and Validation Performance Metrics for BERT and CNN

Epoch	BERT		CNN	
	Train Loss	Test Loss	Train Loss	Test Loss
1	0.5807	0.0748	0.3695	0.1167
2	0.0219	0.0035	0.0966	0.0620
3	0.0039	0.0016	0.0624	0.0441



(a) CNN (AUC: 0.999)

(b) BERT (AUC: 1.000)

Figure 2: ROC Curves showing trade-off between True Positive Rate and False Positive Rate across BERT and CNN.

5 Contribution Scope of Team Members

Sneha: Worked on cleaning data (mainly creating Flow ID Plus parameter) and developing/tuning BERT model. Wrote conclusion and future directions section. Onkita: Assisted with developing the BERT model and tuning hyperparameters. Wrote methodology and experimental results for final paper. Deekshita: Yemmy:

6 Conclusion and Future Directions

The BERT model was able to successfully demonstrate how the text-based approach can be taken to network flow analysis. The BERT model was able to be adjusted to behave perfectly in the small, targeted data set looking at only brute force attempts. The methodology covered preprocessing the mostly numerical aspects of networks flows into more semantic, text-like features that BERT would perform better on, as seen in the 'Flow ID Plus' column, which was the primary encoding used to train BERT. The main aspect of 'Flow ID Plus' that made it successful was contextualizing the information with labels . When 'Flow ID Plus' was used with just numerical values, it was found that the BERT model performed poorly, which was attributed to the model not being able to utilize its language-based pre-training.

Another key part of the BERT model was the weightage of the benign and malicious labels. Within the network packet data, it was seen that most packets were benign and only a select few were malicious, similar to a real-life scenario. It was important for the model not to gain a bias towards a majority trend. For this, a weighted cross-entropy loss function was employed, which assigned a significantly higher weight to the malicious network data class.

In all, with all modifications made the BERT model performed exceptionally, achieving a train and test accuracy of 100%! This result shows that the creation of the 'Flow ID Plus' feature and the transformer architecture are a promising way to detect anomalous behavior

On the other hand, it was seen that the CNN model thrived on the numerical, structured data that naturally appeared in the network data CSV. Unlike BERT, which used text labels to assist the pre-trained model, the CNN model utilized other means of data preprocessing. Categorical identifiers like timestamps and IP addresses were removed, and the remaining features were normalized using

a StandardScaler to ensure uniform data distributions. The deep learning architecture consisted of multiple 1D convolutional layers, max-pooling layers, and dense classification layers. Training over 15 epochs resulted in the best performance of the model, with test accuracy reaching 99.18%. This demonstrates that the statistical flow features, when scaled and normalized, are sufficient for detecting anomalies in network data. While the CNN approach did not achieve a "perfect" score like the BERT model, it shows an alternative to a computational heavier transformer model like BERT and also shows a method that requires less preprocessing of data and works well with raw numerical values.

Future research should prioritize validating these results against diverse and larger-scale datasets. It is more than likely that the above average performance seen in both of these models is due to the fact that they were trained and tested on a small specific dataset, mostly due to computational restrictions working with Google CoLab. In all, the best test of these models would to be to expose them to larger, varied types of network data to see how they generalize under these circumstances

6.1 AI Usage:

AI was an incredibly useful tool that our group did not hesitate to utilize during the course of this project, especially given that Google CoLab had an integrated Gemini agent that could be given context (specific errors, cells, etc.) to solve issues. Our group used Gemini within Google CoLab to lay out the structure and base code of both models. These were then edited and validated by us using knowledge of coding languages, research on models, and looking at other projects online that were attempting similar tasks.

References

Bhattacharya, S., Khanna, A., Ganapaneni, S. & Najana, M. (2024) Attention-Based Deep Learning Frameworks for Network Intrusion Detection: An Empirical Study. *International Journal of Global Innovations and Solutions (IJGIS)*. https://doi.org/10.21428/e90189c8.eb32676c

Manocchio, L.D., Layeghy, S., Lo, W.W., Kulatilleke, G.K., Sarhan, M. & Portmann, M. (2024) FlowTransformer: A transformer framework for flow-based network intrusion detection systems. *Expert Systems with Applications* **241**:122564.

Stein, K., Mahyari, A., Francia III, G. & El-Sheikh, E. (2024) A Transformer-Based Framework for Payload Malware Detection and Classification. arXiv preprint arXiv:2403.18223. Available at: https://arxiv.org/abs/2403.18223